TURING 图灵程序设计丛书

简约之美
软件设计之道

Code Simplicity: The Science of Software Development

[美] Max Kanat-Alexander 著
余晟 译

O'REILLY®
人民邮电出版社
POSTS & TELECOM PRESS

版权信息

书名：简约之美：软件设计之道

作者：Max Kanat-Alexander

译者：余晟

ISBN：978-7-115-30238-0

目录

版权声明

常识——译者序

1776年，美国独立战争爆发，当时北美还有很多民众对“独立”充满怀疑:“北美真的要脱离英国吗”、“新的国家需要怎样组织”,这些今天看来不是问题的问题，并没有清晰明确的答案。就在此时，有位叫托马斯•潘恩的人站了出来，单枪匹马解开了人们心中的疑惑，大大鼓舞了北美民众的独立情绪，而他所依靠的，只是一本名为《常识》的小册子。

《常识》这本小册子说了什么呢？我随便摘录几句：“如果没有人监督，对国王是不能信任的；或者换句话说，渴望保持专制政权的欲念是君主政体的固有弊病”，“独立自主的问题不外乎意味着：究竟是我们将自己制定我们的法律，还是让这个大陆的目前和将来最大的敌人——英王来吩咐我们，除我所喜欢的法律以外不准有任何法律”，“让我们为宪章加冕，从而使世人知道我们是否赞成君主政体，知道北美的法律就是国王”…… 200多年后再读，仍然可以感受到这些文字的力量，所以不难想象，在美国独立战争时，告知民众这些道理，能发挥多么重要的作用。据载，在当时只有200多万人的北美，成年男子几乎人手一册《常识》。不夸张地说，这本书推动了美国建国的进程。

潘恩既不是高明的政治哲学家，也不是熟谙宣传的政客，他的书之所以具有如此大的力量，在我看来，主要原因是他能用朴素平实的语言把道理讲出来，告诉大家“原来是这样的”。换句话说，许多道理其实并不高深，但常识也必须以“常识”的形式表达出来，大家才能听进去。

读者或许会觉得奇怪，一本技术书籍的译者序，为什么要花这么多篇幅介绍历史呢？之所以这么做，是因为我在翻译本书的过程中，数次想到托马斯•潘恩的《常识》。我深刻觉得，在软件开发的各种书籍和资料中，也应当有类似《常识》的文本来告诉大家：道理原来是这样的，就是这样。

我相信，任何一位读者，只要认真看过全书，都会发现《简约之美》其实只强调了几条互相联系的简单道理：软件是必然要变化的，变化是常态；有变化就需要维护，随着时间的推移，维护成本会远远超过初期开发的成本，占据成本的大头；因此，在软件开发中，最重要的是要降低维护成本；维护成本正比于系统的复杂程度，所以要降低维护成本，系

统的设计就应当追求简单清晰。

这根逻辑链条看似简单，其实并非如此。不少有经验的开发人员，似乎对这类“道理”不屑一顾，他们更在意新潮的技术、先进的架构、流行的语言……新出了哪种类库，什么软件新发布了版本，大数据该怎么处理，说起来头头是道，但真刀真枪地写起程序来，往往错漏百出（甚至不自知）。

我曾经见过一套系统，设计和开发这套系统的人几乎用到了.NET的所有高级特性，但95%以上都用错了，结果就是系统层次混乱、类责任混淆、通讯完全不可靠。诡异的是，许多错误都属于“地雷”，如果业务一直维持在原始甚至野蛮的状态，它们几乎不会爆炸。不幸，业务无可避免地要扩展、要增长、要规范，于是地雷一颗接一颗地爆炸，只能投入大量优秀程序员，花比开发多好几倍的精力，去维护和重构系统，才勉强保证了业务的发展。最终，当系统重构告一段落之后回头看，其实所谓的“维护”，也无非是“降低维护成本”，一点点地去掉之前那些花哨的特性，用简单的技术构建清晰的架构，而已。

就我所见，上面这种情况并不罕见，反而非常常见，更糟糕的是，还有许多项目始终不得解脱，被高昂的维护成本死死困住；即便推倒重来，因为设计和开发仍然缺乏对“降低维护成本”的足够重视，导致悲剧重现…… 说起来，这一切的根源都是因为目标不明确，没有考虑且不重视维护成本，也没有考虑设计的简洁清晰。其中的道理不算复杂，但怎么才能让大家明白？这个问题我一直在思考，所以在翻译本书时，才会反复想起托马斯•潘恩的《常识》——软件开发需要常识，软件开发的资料里需要《常识》之类的读本，不需要艰深的道理，也不需要花哨的说辞。潘恩的《常识》用短小的篇幅向广大民众澄清了“北美应该独立，且不需要国王”，我希望《简约之美》也能用短小的篇幅向广大开发人员说明“应当重视软件的维护成本，追求简单清晰的设计”。

——余晟，2012年12月6日

> 余晟，毕业于东北师范大学计算机系，副修中文，非正统型技术爱好者。曾任抓虾网、银杏泰克主力程序员，盛大创新院高级研究员，现任华南某电商公司技术总监。坚信计算机可以无限延伸人的能力，前提是人必须理解计算机的逻辑，所以对任何技术都不应该浅尝辄止，仅仅满足于“会用”。

前言

好程序员和差程序员的区别在于理解能力 。差劲的程序员不理解自己做的事情，优秀的程序员则相反。信不信由你，道理就是这么简单。

写这本书，是为了帮助各位程序员，以适用于各种编程语言、各种项目的广阔视角来理解软件开发。本书以普通人容易理解的方式，讲解了软件开发的科学规律。

如果你是程序员，这些规律能够说明，为什么有些开发方法有效，另一些无效。这些规则也会指引你在日常工作中做出开发决策，帮助你的团队进行高质量的交流，最终制定出合理的计划。

如果你不是程序员，但身在软件行业，仍然可以享受到本书的价值：

- 它既是提供给初级程序员的优秀教材，又包含对高级程序员相当有用的知识；
- 它帮助你更深入地理解软件工程师某些行为的原因，以及软件为何要以某种方式来开发；
- 它帮助你理解优秀的软件工程师做决定的基本原理，让你与开发人员更顺畅地沟通。

理想的状态是，软件行业中的每个人都可以阅读并理解这本书，即便他们没有多少编程经验，甚至母语不是英语也无所谓。如果你已经有相当的技术积累，把握书中的概念会更加容易，但是大部分内容不需要编程经验就能理解。

实际上，本书虽然讲的是软件开发，却没有多少代码。这怎么可能呢？答案是，其中的思想适用于各种软件项目、各种语言。要明白如何运用这些思想，并不需要懂得某一门具体的编程语言。相反，本书中包含了大量的实例和比喻，它们会让你更好地理解所表述的每条原理。

最重要的是，这本书是为了帮助你而写的，希望能助你在软件开发中保持头脑清醒、遵守秩序、写出简洁代码。我希望它读起来是一种享受，

它有助于改善你的生活，你的软件。

排版约定

本书中格式约定如下。

黑体
表示新术语。

等宽字体
用于代码示例，在段落中使用时，表示与程序有关的部分，比如变量或者函数名。

AAA 此图标表示提示、建议或者普通的旁注。

BBB 此图标表示警告或者留意。

致谢

Andy Oram和Jolie Kanat两位编辑为本书作了巨大的贡献。Andy的建议和意见深入且充满智慧；Jolie的坚持和支持促成了本书的最后出版，她为早期手稿所做的大量编辑工作尤其值得感谢。

Rachel Head是本书的文字编辑，做整理和校对的工作，她的才华无与伦比。

还要感谢的是与我在开源社区中一同工作或讨论过问题的程序员——尤其是在Bugzilla项目中共事的几位开发人员，有了你们的帮助，我才有清楚的思维，讲解这些年来真实存在的，活生生的软件系统。

这些年来，我的blog上收到的评论和反馈，帮我确定了本书的形式和内容。在这里要感谢参与其中的所有人，即使你们仅仅给我鼓励，或者是告诉我你读过我的文章。

从个人来说，我尤其要感谢Jevon Milan、Cathy Weaver，以及与他们工作过的所有人。确切地说，有了他们，我才能写出这本书。最后，要向我的朋友Ron致敬，没有他，这本书根本不可能出现。

使用示例代码

让我们助你一臂之力。也许你要在自己的程序或文档中用到本书中的代码。除非大段大段地使用，否则不必与我们联系取得授权。例如，无需请求许可，就可以用本书中的几段代码写成一个程序。但是销售或者发布O’Reilly 图书中代码的光盘则必须事先获得授权。引用书中的代码来回答问题也无需授权。将大段的示例代码整合到你自己的产品文档中则必须经过许可。

我们非常希望你能标明出处，但并不强求。出处一般含有书名、作者、出版商和ISBN，例如“Code Simplicity：The Science of SoftwareDevelopment by Max Kanat-Alexander（O’Reilly，2012） 版权所有，978-1-4493-1389-0”。如果有关于使用代码的未尽事宜， 可以随时与我们取得联系，permissions@oreilly.com。

Safari®在线图书

Safari 在线图书是应需而变的数字图书馆。它能够让你非常轻松地搜索 7500 多种技术性和创新性参考书以及视频，以便快速地找到需要的答案。

订阅后就可以访问在线图书馆内的所有页面和视频。可以在手机或其他移动设备上阅读，还能在新书上市之前抢先阅读，也能够看到还在创作中的书稿并向作者反馈意见。复制粘贴代码示例、放入收藏夹、下载部分章节、标记关键点、做笔记甚至打印页面等有用的功能可以节省大量时间。

这本书（英文版）也在其中。欲访问本书英文版的电子版，或者由 O’Reilly 或其他出版社出版的相关图书，请到 http://my.safaribooksonline.com 免费注册。

我们的联系方式

请把对本书的评论和问题发给出版社。

美国：

O’Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2 号成铭大厦C 座807 室（100035）

奥莱利技术咨询（北京）有限公司

O’Reilly 的每一本书都有专属网页，你可以在那儿找到关于本书的相 关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：
http://www.oreilly.com/catalog/9781449313890

中文书： http://www.oreilly.com.cn/index.php?
func=book&isbn=9787115302380

对于本书的评论和技术性的问题，请发送电子邮件到：
bookquestions@oreilly.com

关于本书的更多信息、会议、资源中心和网络，请访问以下网站：

http://www.oreilly.com

http://www.oreilly.com.cn

第1章 引言

计算机带来了社会的剧变。因为有了它，我们可以用更少的人，干更多的事情。这就是计算机的价值——它可以干很多的事情，而且速度相当快。

这挺棒的。

但是，计算机会出问题，而且总出问题。如果你家里其他东西出问题有计算机那么频繁，你多半会退货。生活在现代的大多数人，每天至少会遇到一次系统崩溃或者程序错误。

这就没那么棒了。

1.1 计算机出了什么问题？

为什么计算机这么容易出问题？如果是软件的问题，那么有而且只有一个原因：编程写得太糟糕。有些人怪罪管理，有些人怪罪客户，但是调查发现，问题的根源通常都在于编程。

那么，“编程写得太糟糕”是什么意思？这是个很模糊的说法。通常来说，程序员都是很聪明、很理智的人，他们怎么会编出“糟糕”的程序呢？

说穿了，这一切都与复杂性有关。

今天，计算机大概是我们能生产的最复杂的设备了。它每秒钟可以计算数十亿次，它内部数以亿计的电子元件必须精密协调，整台计算机才可以正常运行。

计算机上跑着的程序同样复杂。举例来说，微软的Windows 2000还在开发时，就可算有史以来规模最大的软件了，它包含3000万行代码，这大概相当于2亿字——是《不列颠百科全书》字数的5倍。

程序的复杂性问题可能更加麻烦，因为程序里没有摸得着的东西。程序出问题的时候，你也找不到什么实实在在的东西，打开瞧瞧里面发生了什么。程序完全是抽象的，非常难处理。其实，常见的计算机程序就已经足够复杂，没有人可以从头到尾理解所有代码是如何工作的。程序越大，越是如此。

这样说来，编程就成了把复杂问题化解为简单问题的劳动。否则，一旦程序达到某种复杂程度，就没有人可以理解了。程序中复杂的部分必须以某种简单方式组织起来，这样，不需要神那样强大的思维，普通程序员也可以开发出来。

这就是编程所要用到的艺术和才能——化繁为简。

“差程序员”是不会化繁未简的。他们总以为用编程语言（这东西本来就够复杂了）写出“能跑通”的程序，就已经化解了复杂性，而没有考虑降低其他程序员需要面对的复杂性。

大概就是这么一回事。

设想一下，为了把钉子钉到地板上，工程师发明了一台机器，上面有皮带轮，有绳子，还有磁铁。你多半会觉得这很荒唐。

再设想，有人告诉你说：“我需要一段代码，它能用在任何程序的任何地方，能够通过任何介质，实现两台计算机之间的通信。”这个问题无疑很繁为简。所以，有些程序员（大多数程序员）在这种情况下给出的解法，就像是一台装备了皮带轮、绳子、磁铁的机器，其他人当然很难看得懂。并不是这些程序员缺少理性，他们的脑子也没有进水。他们面对的问题确实困难，设定的期限也很紧，他们能做的就是这些。在这些程序员看来，写出来的东西是能用的，它符合要求。这就是他们的老板需要的，看来也应该是客户需要的。

不过，这些程序员毕竟没做到化繁为简。完工之后，他们把结果交给其他程序员，其他程序员又会在这之上继续增添复杂性，完成自己的工作。程序员对化解复杂性考虑得越少，程序就越难懂。

于是程序变得无比复杂，最终没办法找出其中的各种问题。喷气式飞机差不多也有这么复杂，但它们的造价是几百万甚至几十亿美元，而且仔细排查过错误。大多数软件的售价只有50～100美元。价钱这么低，没有人有足够的时间和资源，在几乎无限复杂的系统里找到所有的问题。

所以，“好程序员”应当竭尽全力，把程序写得让其他程序员容易理解。因为他写的东西都很好懂，所以要找出bug是相当容易的。

这个关于简单性的想法有时被误解为，程序不应当包含太多代码，或者是不应当使用先进技术。这么想是不对的。有时候，大量的代码也可以带来简单，只不过增加了阅读和编写的工作量而已，这是完全正常的。你只要保证，那些大段的代码提供了化解复杂性所必须的简短注释，就足够了。同样，通常来说，更先进的技术只会让事情更简单，只是一开始你得学习，所以整个过程可能没那么简单。

有些人相信，把程序写得简单所花的时间，要比写“能用就好”的程序更多。其实，花更多的时间把程序写简单，相比一开始随意拼凑些代码再花大量的时间去理解，要快得多。这个问题说起来轻巧，显得轻描淡写，其实软件开发的历史教训很多，诸多事例已经反复证明了这一点。许多大型程序的开发之所以会停滞数年，就是因为一开始没有做好，结

果必须等上这么长的时间，才能给之前开发出来的怪物加上新功能。

正因为如此，计算机经常出问题——因为大多数程序都有这个问题，许多程序员在写程序时并没有化解复杂性。是的，这么做很难。但是如果程序员做不到这一点，设计出的系统过于复杂、经常出问题，用户在使用的过程中就会经受无穷无尽的折磨。这么一比，把程序写简单所费的工夫实在算不了什么。

1.2 程序究竟是什么？

大多数人说的“计算机程序”，其实有完全不同的定义：

（1）给计算机的一系列指令

（2）计算机依据指令进行的操作

第一种定义是程序员写程序时所用的。第二种定义是使用程序的普通用户所用的。程序员命令计算机：在屏幕上显示一头猪。这就是第一种定义，它包含若干指令。计算机接收到指令之后，会控制电信号，在屏幕上显示一头猪。这是后一种定义，即计算机执行的操作。程序员和用户都会说自己在和“计算机程序”打交道，但是他们的用法是很不一样的。程序员面对的是字母和符号，用户看到的是最终结果——计算机执行的操作。

所以，计算机程序其实是这两者的混合体：程序员的指令、计算机执行的操作。编写指令的最终结果就是让计算机执行那些操作——如果不需要执行操作，就没必要去写代码了。这就好像在生活里，你列了一张购物单（相当于指令），告诉自己该买哪些东西。如果你只是列了单子，但没去商店，单子就没有任何意义。指令必须得到实际的结果。

但是，列购物单和写程序有显著的区别。如果购物单列得很乱，只不过会降低买东西的速度。但是如果程序写得很乱，实现最终的目标就显得尤其困难。为什么呢？因为购物单是简单短小的，用完就可以扔掉。而程序是很复杂很庞大的，你可能还需要维护很多年。所以，同样是没有秩序，购物单只会给你造成一点儿小麻烦，程序却可以给你增添无尽的烦恼。

而且，除软件开发之外，没有任何领域的指令和结果联系得这么紧密。在其他领域，人们先编写指令，然后交给其他人，指令通常要等很长的时间才会执行。比如设计房子，建筑师首先给出指令——也就是蓝图。这份蓝图经很多人的手，过了很长的时间，才能建起真正的房子。所以，房子是大家所有人解读建筑师指令的结果。相反，写程序时，在我们和计算机之间没有任何人。我们让计算机干什么，就会得到怎样的结果；计算机绝对服从命令。结果的质量完全取决于机器的质量、我们想

法的质量，代码的质量。
*